Тема: Використання UI Components в середовищі Cocos2d-x v. 3.x

Мета: Ознайомити з особливостями використання на налаштування компонентів інтерфейсу користувача: labels, buttons, menus, sliders і views

Зміст:

1. Label та шрифти
   - *Label BMFont*
   - *Label TTF*
   - *Label SystemFont*
2. Label ефекти
3. Меню та елементи меню
4. Лямда функції як функція зворотного виклику Меню
5. Buttons
6. CheckBox.
7. LoadingBar
8. ScrollView
9. Slider
10. TextField

Існує багато компонент, які не є елементами гри, але є невід'ємною складовою інтерфейсу. Вони дозволяють здійснювати логістику по структурі гри, формувати інтерактивне спілкування з гравцем, та формують динамічність компонентів. Це компоненти інтерфейсу користувача (**UI Components).**
Віджети!

UI - абревіатура, що символізує користувальницький інтерфейс. Те що знаходиться на екрані. Це включає в себе такі елементи: мітки, кнопки, меню, слайдери та переглядачі (labels, buttons, menus, sliders і views..) Cocos2d-x надає набір віджетів інтерфейсу користувача, що полегшує додавання до проектів та керування цими елементами.

Label та шрифти

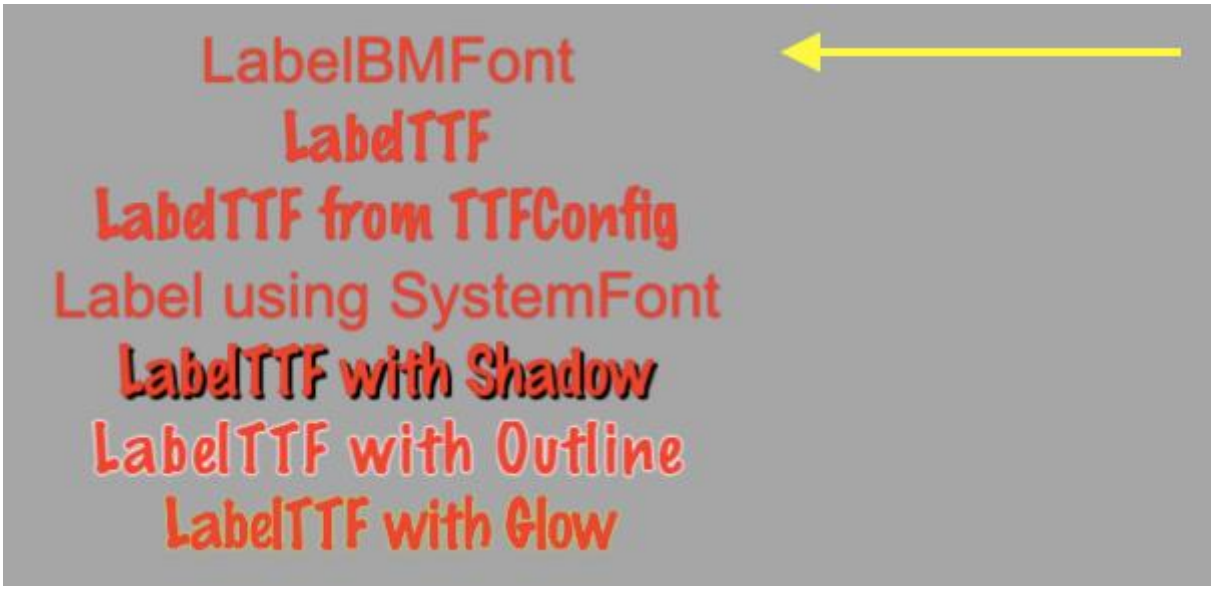
Cocos2d-x має в своєму розпорядженні об'єкт Label, який може створювати написи з використанням справжнього типу, растрового зображення або вбудованого системного шрифту. Цей єдиний клас володіє всіма необхідними методами для ефективного використання елементу Label.

Label BMFont

BMFont - це тип напису, який використовує растровий шрифт. Символи в растровому шрифті складаються з матриці точок. Це дуже швидкий і простий у використанні, але не масштабований, оскільки для цього потрібен окремий шрифт для кожного розміру символу. Кожен символ у елементі є окремим спрайтом. Це означає, що кожен символ може бути повернений, масштабований, тонований, мати іншу точку прикріплення та / або більшість будь-яких інших властивостей спрайту.

Створення мітки BMFont вимагає двох файлів: а .fnt-файлу та представлення зображення кожного символу у форматі .png. Якщо ви використовуєте такий інструмент, як Glyph Designer, ці файли створюються автоматично для вас. Створення об'єкта Label з растрового шрифту:

```
auto myLabel = Label::createWithBMFont("bitmapRed.fnt", "Your Text");
```



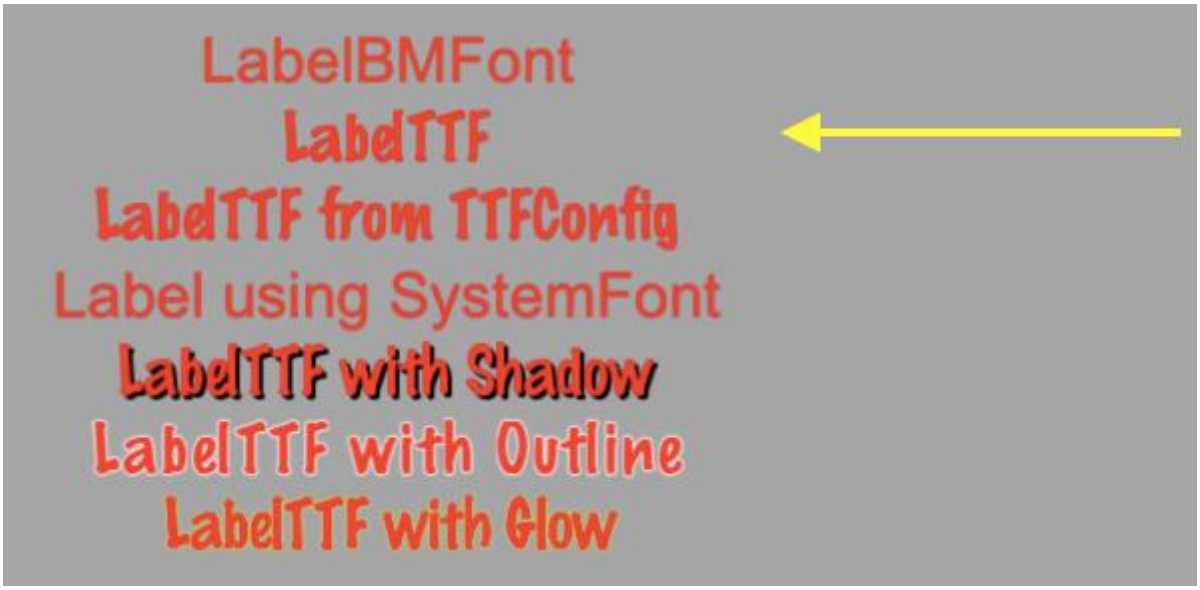
мал. 1

Всі символи в рядковому параметрі повинні бути знайдені у файлі .fnt, інакше вони не будуть відтворені. Якщо в Label буде прописано відсутні символи, то їх попередньо потрібно прописати у файлі .fnt.

Label TTF

Шрифти True Type відрізняються від растрових шрифтів, про які йшла мова вище. У True Type шрифтах відображається контур шрифту. Це зручно, оскільки не потрібно мати окремий файл шрифту для кожного розміру та кольору, який ви можете використовувати. Створення об'єкту Label, який використовує True Type шрифт, легко. Щоб створити його, потрібно вказати ім'я файлу шрифту .ttf, текстовий рядок та розмір. На відміну від BMFont, TTF може робити зміни розміру без необхідності окремих файлів шрифтів. Наприклад, використтання True Type шрифту реалізовується наступною командою:

```
auto myLabel = Label::createWithTTF("Your Text", "Marker Felt.ttf", 24);
```



мал. 2

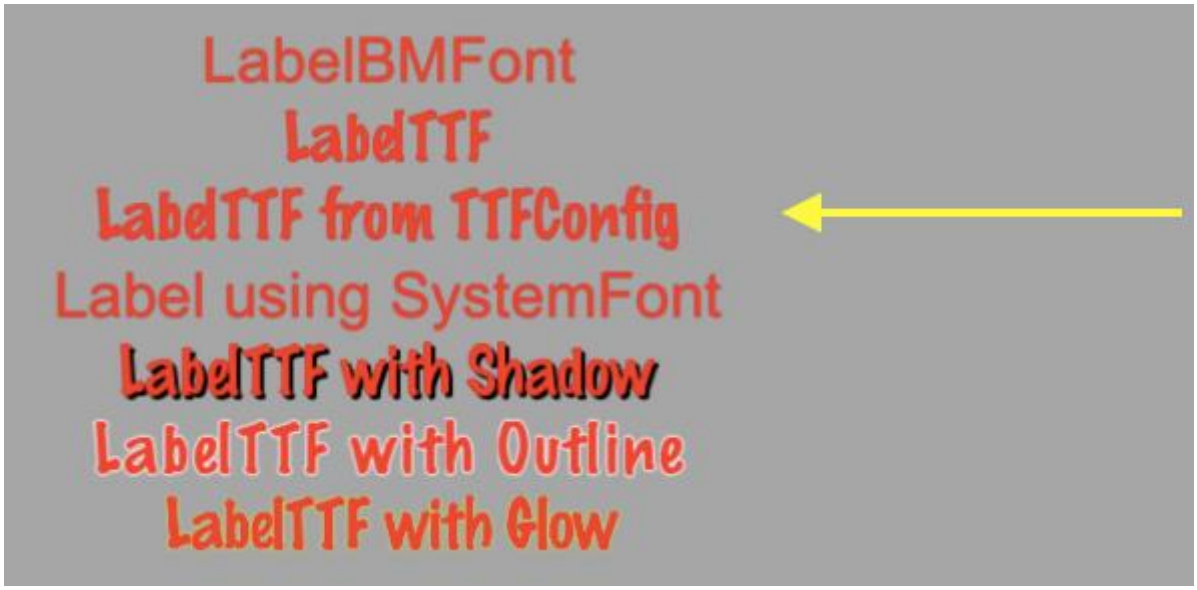
Незважаючи на те, що він є більш гнучким, ніж растровий шрифт, True Type тип шрифту повільніше рендериться і змінює властивості, такі як шрифт, і розмір - це громіздка операція.

Якщо вам потрібні кілька об'єктів Label з True Type шрифту, який має всі ті самі властивості, ви можете створити об'єкт TTFConfig, щоб керувати ними. Об'єкт TTFConfig дозволяє встановлювати властивості, які мають всі спільні мітки. Ви можете подумати про це, як про рецепт, де всі об'єкти Label використовуватимуть ті ж самі інгредієнти.

Ви можете створити об'єкт TTFConfig для своїх об'єктів Label таким чином:

```
// створення об'єкту TTFConfig
        TTFConfig labelConfig;
        labelConfig.fontFilePath = "myFont.ttf";
        labelConfig.fontSize = 16;
        labelConfig.glyphs = GlyphCollection::DYNAMIC;
        labelConfig.outlineSize = 0;
        labelConfig.customGlyphs = nullptr;
        labelConfig.distanceFieldEnabled = false;

        // створення TTF Label на основі TTFConfig об'єкту.
        auto myLabel = Label::createWithTTF(labelConfig, "My Label Text");
```



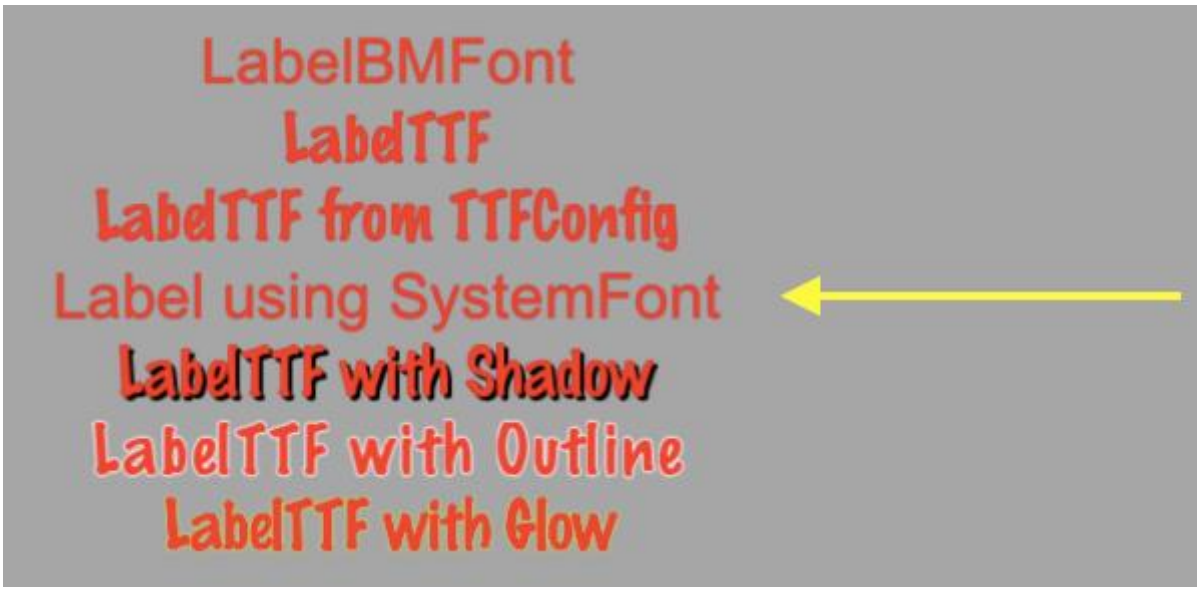
мал. 3

`TTFConfig` може використовувати китайські, японські та корейські символи.

Label SystemFont

SystemFont - це тип Label, який використовує системний шрифт за замовчуванням та розмір шрифту. Цей шрифт означає, що його властивості не змінюються. Створення мітки SystemFont:

```
auto myLabel = Label::createWithSystemFont("My Label Text", "Arial", 16);
```



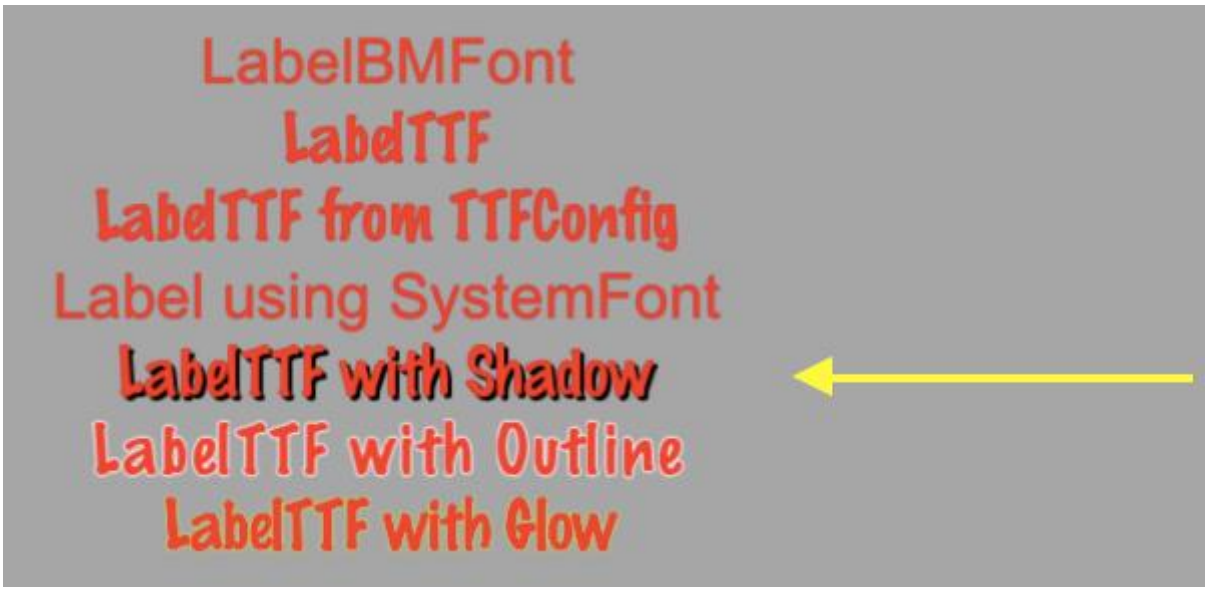
мал. 4

Label ефекти

Після того як у вас є об'єкти Label на екрані, до них можна застосувати деякі стандартні ефекти. Використання ефектів до стандартних шрифтів дозволить суттєво змінити їх вигляд, та дозволить обійти потребу у створені нових шрифтів. Правда не всі об'єкти Label підтримують всі ефекти. Деякі ефекти включають тінь, контур і світло. Ви можете застосувати один або кілька ефектів до об'єкта Label.
Мітка з ефектом тіні:

```
auto myLabel = Label::createWithTTF("myFont.ttf", "My Label Text", 16);

    // ефект тіні
    myLabel->enableShadow();
```



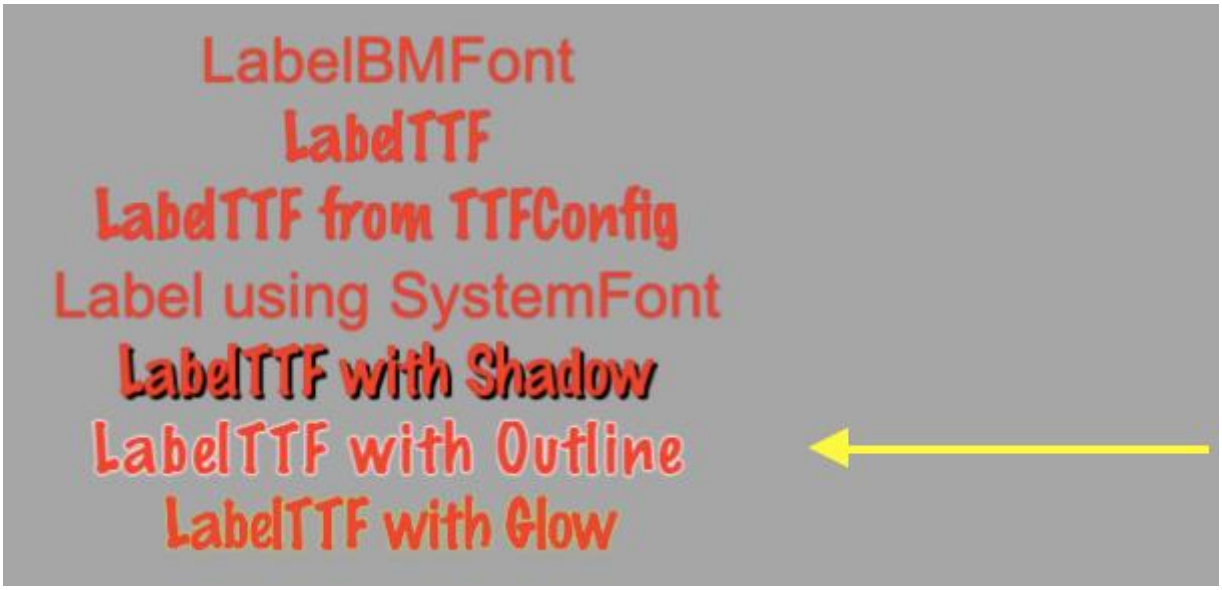
мал. 5

Label з ефектом зовнішньої лінії:

```
auto myLabel = Label::createWithTTF("myFont.ttf", "My Label Text", 16);

    // ефект зовнішньої лінії
    myLabel->enableOutline(Color4B::WHITE, 1));
```

Даний ефект може використовуватись тільки до ttf шрифтів, і потрібно вказати колір та товщину лінії



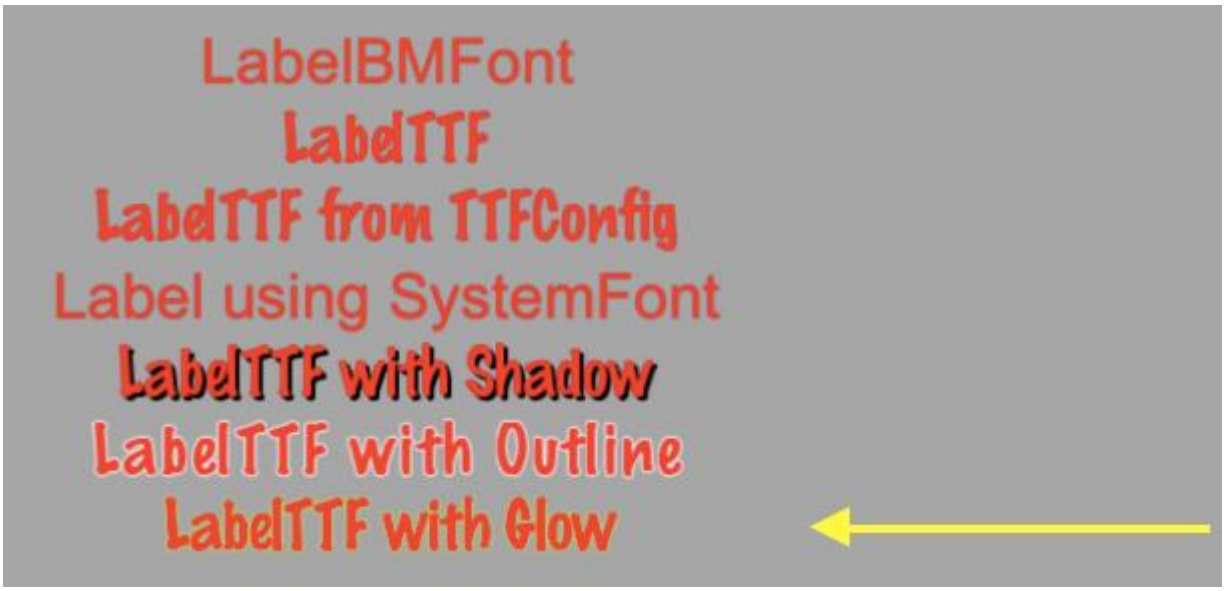
мал. 6

Label з ефектом підсвічення:

```
auto myLabel = Label::createWithTTF("myFont.ttf", "My Label Text", 16);

    // ефект пісвічення
    myLabel->enableGlow(Color4B::YELLOW);
```

Ефект підсвічення використовується виключно до до ttf шрифтів, також потрібно вказати колір підсвічення.



мал. 7

Меню та елементи меню

Ми всі, мабуть, знайомі з тим, що таке меню. Ми бачимо це в кожному застосуванні, яке ми використовуємо. У вашій грі ви б, напевно, скористалися

об'єктом Menu для навігації через параметри гри. Меню часто містить кнопки Play, Quit, Settings і About, але можуть також містити інші об'єкти Menu для вкладеної системи меню. Об'єкт меню - це особливий тип об'єкта Node. Ви можете створити пустий об'єкт меню як власник місця для ваших пунктів меню:

```
auto myMenu = Menu::create();
```

Як було сказано вище пункти початок, вихід, налаштувань елемент про авторів, це елементи меню. Меню без елементів меню не має сенсу. Cocos2d-x пропонує різні способи створення елементів меню, включаючи використання об'єкта Label або визначення зображення для відображення. Пункти меню, як правило, мають два можливих стани, звичайний і вибраний стан. Коли ви торкаєтесь або клацаєте на елементі меню, спрацьовує зворотний виклик. Ви можете придумати це як ланцюгову реакцію. Ви торкаєтесь або клацаєте на елементі меню, і він виконує вказаний вами код. У меню може мати від одного елемента до кількох елементів.

```
        // створення меню з одним елементом
        // елемент меню створюється з спеціальних зображень
auto closeItem = MenuItemImage::create("CloseNormal.png", "CloseSelected.png",
                CC_CALLBACK_1(HelloWorld::menuCloseCallback, this));

        auto menu = Menu::create(closeItem, NULL);
        this->addChild(menu, 1);
```

Меню також може бути створене за допомогою вектора об'єктів MenuItem:

```
// створення меню з вектора менюелементів
        Vector<MenuItem*> MenuItems;

        auto closeItem = MenuItemImage::create("CloseNormal.png",
"CloseSelected.png",
                CC_CALLBACK_1(HelloWorld::menuCloseCallback, this));

        MenuItems.pushBack(closeItem);

        /* повторити цю процедуру стільки раз скільки потрібно, для кожного елемента
меню */

        auto menu = Menu::createWithArray(MenuItems);
        this->addChild(menu, 1);
```

Лямда функції як функція зворотного виклику Меню

Вище ми просто дізналися, що, коли ви натискаєте пункт меню, він викликає зворотний виклик. C ++ 11 пропонує лямбда-функції, і тому Cocos2d-x цілком використовує їх! Функція лямбда - це функція, яку ви пишете в рядку у

вихідному коді. Lambda також оцінюється під час виконання, замість часу компіляції.
Проста лямбда:

```
// створення простої лямбда функції
        auto func = []() { cout << "Hello World"; };

        // виклик функції десь в коді
        func();
```

Як відомо з теорії вище, що, коли ви натискаєте пункт меню, викликається функція зворотнього виклику. C ++ 11 пропонує лямбда-функції, і тому Cocos2d-x цілком використовує їх! Функція лямбда - це функція, яку ви пишете в рядку у вихідному коді. Lambda також оцінюється під час виконання, замість часу компіляції.
Проста лямбда функція:

```
// створення простої лямбда функції
        auto func = []() { cout << "Hello World"; };

        // виклик функції
        func();
```

Використання лямбда функції в якості функції зворотнього виклику при задані пункту меню:

```
        auto closeItem = MenuItemImage::create("CloseNormal.png",
"CloseSelected.png",
                [&](Ref* sender) {
                // код функції
        });
```

Buttons

Buttons це один з найрозповсюджених UI компонентів, ви можете використовувати кнопку, щоб змінити сюжети або додати об'єкти Sprite у свою гру. Кнопка перехоплює сенсорну подію і викликає попередньо визначену функцію зворотної дії. Кнопка має звичайний і активний стан. Вигляд кнопки може змінюватися залежно від її стану. Щоб використовувати клас Кнопки та інші частини графічного інтерфейсу, необхідно включити файл CocosGUI.h. Додамо наступний рядок коду в HelloWorldScene.cpp:

Створення кнопки та визначення її дії, реалізується наступним кодом:

```
#include "ui/CocosGUI.h"
... ... ... ... ... ...
auto button = ui::Button::create("3.png", "2.png", "1.png");
        //напис на кнопці
```

```
        button->setTitleText("Button Text");
        // позиціонування кнопки
        button->setPosition(Vec2(50,50));
        // встановлення події коли проводяться дії з кнопкою
button->addTouchEventListener([&](Ref* sender, ui::Widget::TouchEventType type) {
                switch (type)
                {
                case ui::Widget::TouchEventType::BEGAN:
                        CCLOG("touch began");
                        break;
                case ui::Widget::TouchEventType::MOVED:
                        CCLOG("touch moved");
                        break;
                case ui::Widget::TouchEventType::ENDED:
                        CCLOG("touch ended");
                        break;
                case ui::Widget::TouchEventType::CANCELED:
                        CCLOG("touch canceled");
                        break;
                default:
                        break;
                }
        });
        scene->addChild(button);
```

`ui::Button::create` – має три параметри normal_image - selected_image - disabled_image

Файл CocosGUI.h, визначає необхідні класи для елементів UI. Далі, зверніть увагу що ці класи мають власний простір імен, таких як "`cocos2d :: ui`." Легко створити екземпляр класу Кнопки. Вам потрібно лише вказати спрайт ім'я файлу Крім того, ви можете створити функцію зворотного виклику як лямбда-вираз за допомогою метод **`addTouchEventListener`**. Ця функція має два параметри. Перший параметр це кнопка, яка була натиснута. Другий параметр - це сенсорний статус. Торкнутися статуси складаються з чотирьох типів. **`TouchEventType :: BEGAN`** - статус в той момент, коли кнопка натиснута. **`TouchEventType :: MOVE`** - це тип події, який виникає при переміщенні пальцем після натискання. **`TouchEventType :: ENDED`** - це подія, яка виникає на момент відпустіть палець на екрані. **`TouchEventType :: CANCELED`** - це подія що виникає, коли ви відпускаєте палець поза кнопкою.

Також можна запрограмувати кнопку на виклик функції-користувача.

Нижчеописаний код реалізує виклик іншої сцени через функцію `GoToGameScene`

```
auto button = ui::Button::create("3.png", "2.png", "1.png");
        //напис на кнопці
        button->setTitleText("Button Text");
        // позиціонування кнопки
        button->setPosition(Vec2(50, 50));
```

```
        button->addTouchEventListener(CC_CALLBACK_1(MainMenu::GoToGameScene,
this));
        this->addChild(button);
```

CheckBox

Ми всі використовуємо елемент для заповнення прапорців на паперових формах, таких як заявки на роботу та договори оренди. Ви також можете мати прапорці у ваших іграх. Можливо, ви хочете мати можливість вашого гравця зробити простий варіант так чи ні. Чекбокси працюють в стані бінарного вибору – активний, не активний (1,0). Чекбокс дозволяє користувачеві зробити такий вибір. Є 5 різних станів, які може мати прапорець: нормальний, нормальний – натиснутий, активний, галочка по замовчування, галочка активна. Відповідно до кожного стану потрібно своє зображення. Створити CheckBox дуже просто:

```
auto checkbox = ui::CheckBox::create("check_box_normal.png ",
                "check_box_normal_press.png ",
                "check_box_active.png ",
                " check_box_normal_disable.png ",
                " check_box_active_disable.png ");

        checkbox->addTouchEventListener([&](Ref* sender, ui::Widget::TouchEventType
type) {
                switch (type)
                {
                case ui::Widget::TouchEventType::BEGAN:
                        break;
                case ui::Widget::TouchEventType::ENDED:
                        CCLOG("touch moved");
                        break;
                default:
                        break;
                }
        });
```

Як ви можете бачити у наведеному вище прикладі, ми вказуємо зображення .png для кожного з можливих станів, в яких може перебувати прапорець. Оскільки існує 5 можливих станів, якими може бути CheckBox, то це 5 зображень, по одному для кожного з них

 check_box_normal.png

 check_box_normal_press.png

 check_box_active.png

check_box_normal_disable.png

check_box_active_disable.png

На екрані елемент CheckBox може мати наступний вигляд:

мал. 8

Також можна отримати статус прапорця за допомогою методу `isSelected`.

```
if(checkbox->isSelected()) {
		CCLOG("selected checkbox");
	}
	else {
		CCLOG("unselected checkbox");
	}
```

Ви також можете змінити статус прапорця за допомогою методу `setSelected`.

```
checkbox->setSelected(true);
```

Проаналізувати стан прапорця та виконатит відповідну дію, можна також за допомогою методу `addEventListener`.

```
checkbox->addEventListener([](Ref* sender, ui::CheckBox::EventType
		type) {
		switch (type) {
```

```
            case ui::CheckBox::EventType::SELECTED:
                CCLOG("selected checkbox");
                break;
            case ui::CheckBox::EventType::UNSELECTED:
                CCLOG("unselected checkbox");
                break;
            default:
                break;
            }
    });
```

LoadingBar

Коли відбувається якийсь процес чи завантажуєте щось як правило збоку користувача виникає думка зависання системи, можна показати цей процес користувачеві, вказуючи інтенсивність процесу тай частини процесу, який вже відбувся.

Щоб показати такі прогреси, Cocos2d-x має клас `LoadingBar`.
По-перше, ми повинні підготувати зображення до панелі виконання. Це зображення називається панеллю завантаження.

мал. 9

Він створює екземпляр панелі завантаження, задаючи зображення панелі завантаження. Далі він встановлюється на 0% за допомогою методу setPercent. Нарешті, для того, щоб просунути бар від 0% до 100% на 1% при 0,1 с, ми будемо використовувати метод розкладу наступним чином:

```
auto loadingbar = ui::LoadingBar::create("pb.png");
    loadingbar->setPosition(Vec2(100,100));
    loadingbar->setPercent(0);
    this->addChild(loadingbar);
    this->schedule([=](float delta) {
        float percent = loadingbar->getPercent();
        percent++;
        loadingbar->setPercent(percent);
        if (percent >= 100.0f) {
            this->unschedule("updateLoadingBar");
        }
    }, 0.1f, "updateLoadingBar");
```

Ви повинні вказати одне зображення як зображення на панелі завантаження, щоб створити екземпляр `LoadingBar` клас. Ви можете встановити відсоток завантажувальної панелі, використовуючи `setPercent` метод Крім того, ви можете отримати його відсоток, використовуючи метод `getPercent`.

За замовчуванням панель завантаження просувається праворуч. Ви можете змінити цей напрямок використовуючи метод `setDirection`.

```
loadingbar-> setDirection (ui :: LoadingBar :: Direction :: RIGHT);
```

Коли ви встановлюєте `ui :: LoadingBar :: Direction :: RIGHT`, початкова позиція `LoadingBar` завантаження - правий край. Потім, панель завантаження буде прогресувати у лівому напрямку.

Використовуючи метод `setPercent` можна встановлювати прогрес вручну

```
loadingbar->setPercent(50);
```

ScrollView

Коли у вашій грі відображається величезна карта, потрібно забезпечити прокрутку. Зображеня може бути прокручене проведенням пальця і що забезпечить переміщення до краю області.

Вигляд прокрутки можна створити за допомогою наступного коду:

```
auto scrollView = ui::ScrollView::create();
        scrollView->setPosition(Vec2());
        scrollView->setDirection(ui::ScrollView::Direction::BOTH);
        scrollView->setBounceEnabled(true);
        this->addChild(scrollView);
        auto sprite = Sprite::create("HelloWorld.png");
        sprite->setScale(2.0f);
        sprite->setPosition(sprite->getBoundingBox().size / 2);
        scrollView->addChild(sprite);
        scrollView->setInnerContainerSize(sprite->getBoundingBox().size);
        scrollView->setContentSize(sprite->getContentSize());
```

Проаналізуємо код:

1. Створено екземпляр класу `ScrollView`, використовуючи метод створення без аргументів
2. Задано напрямок перегляду прокрутки, використовуючи метод `setDirection`. У цьому випадку, ми хочемо прокрутити вгору і вниз, а ліворуч і праворуч, так що вам слід встановити `ui :: ScrollView :: Direction :: BOTH`. Це означає, що ми можемо прокручувати як вертикально так і горизонтально. Якщо ви хочете прокрутити вгору і вниз, встановіть `ui :: ScrollView :: Direction::VERTICAL`. Якщо ви хочете прокрутити ліворуч і праворуч, ви встановите `ui :: ScrollView :: Direction::HORIZONTAL`.
3. Якщо потрібно підстрибувати, коли зображення прокручується на краю області, вам слід встановити true використовуючи метод `setBounceEnabled`.

4. Потрібно вказати розмір вмісту в режимі прокрутки, використовуючи метод `setInnerContainerSize`. У цьому випадку ми вказуємо подвійний розмір HelloWorld.png в методі `setInnerContainerSize`
5. Потрібно вказати розмір перегляду прокрутки, використовуючи `setContentSize` метод У цьому випадку задано оригінальний розмір `HelloWorld.png` за допомогою метод `setContentSize`.

Slider

Слайдер буде використовуватися для таких завдань, як зміна гучність звуку або музики, заміни параметрів ігрових персонажів. Cocos2d-x має для нього клас Slider. Якщо ми використовуємо цей клас, ми повинні передбачити як мінімум два зображення, це:

Background слайдеру (sliderTrack)

мал. 10

Зображення яке буде рухатись по слайдеру (sliderThumb)

мал. 11

Сам екземпляр слайдера можна створити наступною командою:

```
auto slider = ui::Slider::create("res/sliderTrack.png",        "res/sliderThumb.png");
```

Проте задавати властивості слайдера можна і окремими методами:

```
auto slider = Slider::create();
slider->loadBarTexture("Slider_Back.png"); // sliderTrack
slider->loadSlidBallTextures("SliderNode_Normal.png",          "SliderNode_Press.png",
"SliderNode_Disable.png"); // вигляд sliderThumb
slider->loadProgressBarTexture("Slider_PressBar.png"); // вигляд активного слайдеру
```

Програмування дії слайдера (реакції слайдера на зміни) відбувається за тимиж принципами що і в попередніх UI компонентів:

```
slider->addTouchEventListener([&](Ref* sender, Widget::TouchEventType type) {
    switch (type)
    {
    case ui::Widget::TouchEventType::BEGAN:
        break;
    case ui::Widget::TouchEventType::ENDED:
```

```
                CCLOG("slider move");
                break;
        default:
                break;
        }
});
```

Для реагування слайдера на положення можна використати наступний код

```
slider->addEventListener([](Ref* sender, ui::Slider::EventType
                type) {
                auto slider = dynamic_cast<ui::Slider*>(sender);
                if (type == ui::Slider::EventType::ON_PERCENTAGE_CHANGED) {
                        CCLOG("percentage = %d", slider->getPercent());
                }
        });
```

TextField

Інколи буває необхідність ввести якісь дані у грі, це можуть бути параметри вузлів, чи навіть нікнейм гравця. Для введення тексту, використовується клас `TextField`.

Для створення текстового поля, потрібно вказати текст початкового заповнення, назву шрифту та розмір шрифту. Потім, встановлюється функція зворотного виклику, використовуючи `addEventListener`. У функції зворотного виклику можна отримаєте текст, який гравець ввів у `textField`.

Створити `textField`, можна використовуючи наступний код:

```
auto textField = ui::TextField::create("Enter your name", "Arial",
                30);
        textField->setPosition(Vec2(300, 300));
        this->addChild(textField);
        textField->addEventListener([](Ref* sender,
                ui::TextField::EventType type) {
                auto textField = dynamic_cast<ui::TextField*>(sender);
                switch (type) {
                case ui::TextField::EventType::ATTACH_WITH_IME:
                        CCLOG("displayed keyboard");
                        break;
                case ui::TextField::EventType::DETACH_WITH_IME:
                        CCLOG("dismissed keyboard");
                        break;
                case ui::TextField::EventType::INSERT_TEXT:
                        CCLOG("inserted text : %s",
                                textField->getString().c_str());
                        break;
                case ui::TextField::EventType::DELETE_BACKWARD:
                        CCLOG("deleted backward");
```

```
                break;
            default:
                break;
            }
    });
```

Розберемо код:
1. Створюється екземпляр класу `TextField`. Перший аргумент – текст по замовчуванню, другий аргумент - це назва шрифту. Ви можете вказати лише True Type шрифти. Третім аргументом є розмір шрифту.
2. Ви можете отримати подію за допомогою методу `addEventListener`. Наступний список надає назви подій та їх описи

ATTACH_WITH_IME	З'явиться клавіатура.
DETACH_WITH_IME	Клавіатура зникне.
INSERT_TEXT	Текст був введений. Ви можете отримати рядок за допомогою методу getString.
DELETE_BACKWARD	Текст був знищений.

Коли гравець вводить пароль, ви повинні сховати його, використовуючи `setPasswordEnable` метод

```
textField->setPasswordEnabled(true);
```

Коли потрібно обмежити максимальну кількість символів введених в поле, можна скористатись методом `setMaxLength`

```
textField->setMaxLength(10);
```

Список літературних джерел по темі:

1. Siddharth Shekar Learning Cocos2d-x Game Development // 2014 Packt Publishing, p.266, ISBN 978-1-78398-826-6
2. Raydelto Hernandez Building Android Games with Cocos2d-x // 2015 Packt Publishing, p. 160, ISBN 978-1-78528-383-3
3. Roger Engelbert Cocos2d-x by Example Beginner's Guide Second Edition // 2015 Packt Publishing, p. 270, ISBN 978-1-78528-885-2
4. Akihiro Matsuura Cocos2d-x Cookbook // 2015 Packt Publishing, p. 255, ISBN 978-1-78328-475-7
5. Karan Sequeira Cocos2d-x Game Development Blueprints // 2015 Packt Publishing, p. 392, ISBN 978-1-78398-526-5
6. Frahaan Hussain, Arutosh Gurung, Gareth Jones Cocos2d-x Game Development Essentials // 2014 Packt Publishing, p. 392, ISBN 978-1-78398-786-3